JavaServer Pages

Kiszolgálóoldali fejlesztői és futtató környezet kialakítása Linux-rendszeren.

A ma készülő alkalmazások jelentős része a többrétegű modell elve alapján készül. Ez a tervezési mód annyira előretört, hogy mostanában még a régi alkalmazások is e modell kapcsán kapják meg az „egyrétegű”, illetve a „vastag ügyfél” nevet. Természetesen ebben a felfogásban nincs semmi új, hiszen a nagyobb programokat mindig is úgy tervezték, hogy azok függőleges és vízszintes tagolású programegységekből álljanak. Erre klasszikus példa a TCP/IP protokoll, ahol a legismertebb megvalósításnál függőlegesen négy réteg van, ugyanakkor például az IP-réteg is több együttműködő részből áll (ICMP, IP stb.). Itt a *réteg* szó arra is utal, hogy az egymással kapcsolatban álló programelemek hány különálló futáskörnyezetre bonthatók. Amennyiben alaposan megvizsgáljuk a Unix-, vagy a Windows-rendszereket, akkor belátható, hogy ez az elgondolás már nagyon régi (lásd a `.so` és `.dll` fájlokat mint kiszolgálókat). Egy háromrétegű alkalmazás jellemző kialakítása a következő: adatbáziskezelő (AB) réteg – üzleti logika (ÜL avagy lényegi rész) réteg – felhasználói felület (FF avagy megjelenítési) réteg. Itt az adatbázisréteg és az üzleti logika, valamint az üzleti logika és a felhasználói felület kapcsolata elképzelhető egy-egy kiszolgáló-ügyfél kapcsolatként is.

Az üzleti logika nagyjából állandó részét megvalósító alapprogramot alkalmazáskiszolgálónak nevezzük, ennek számos esetben a legfontosabb része egy webkiszolgáló (HTTP-kiszolgáló), bár lényeges kiemelni, hogy a CORBA, DCOM, RMI és más módszerek önmagukban is hatékony üzleti logika kialakítását teszik lehetővé. Alkalmazásunk szolgáltatásait mégis érdemes lehet egy webkiszolgáló köré építeni, hiszen a HTTP protokoll további lehetőséget is biztosít számunkra. Miért? A válasz a programozáselmélet típusfogalmával való összehasonlítással érthető meg.

A TCP szállítási réteg, illetve annak Socket vagy TLI (Transport Layer Interface) API-ja csak egy típusnélküli, nyers bájtfolyamot bocsát a rendelkezésünkre, ami még általában alacsony szintű eszköz azok számára, akik például internetes áruházat szeretnének kialakítani. Ezzel szemben a HTTP és a hozzá kapcsolódó társprotokollok és eszközök (például: a MIME) típusos fogalmak, hiszen a rétegek között teljes objektumok (applet, kép, XML stb.) is közlekedhetnek, amik ismerik saját belső ábrázolásaikat, állapotaikat, típusműveleteiket (metódusok). A HTTP kiszolgáló önmagában nem jelent teljes megoldást, viszont jól együttműködhet más programokkal (CORBA, RMI, Servlet, biztonság stb.). Ezt a felépítést nevezzük alkalmazáskiszolgálónak.

A vékony felhasználói réteg gyakran egy HTML-, XML-, illetve Java-alapú (esetleg ActiveX-alapú) vékony ügyfélalkalmazás lehet. Az adatbázisréteg pedig már mindenki számára ismert, hiszen az Oracle, DB/2 stb. programok jól érzékeltetik, hogy miket is kell e rétegben megvalósítani, illetve milyenek azok a hálózati protokollok (pl.: Net *8), amelyek az adatbázis-szolgáltatások hálózaton keresztüli elérését teszik lehetővé.

A webes alkalmazások fejlődéstörténete

A Microsoft Visual InterDev vagy a SUN JSP-vel foglalkozó írásai szerint – a webes alkalmazások fejlődését nyomon követve – három nagy korszakot különíthetünk el:

- *Az első nemzedékbeli webkiszolgálók korszaka.* Ezen megoldásokra a statikus HTML-oldalak és az azokba beágyazott grafikák (*.jpg, *.gif), hangok, később a mozgófilmek voltak a jellemzők. Ezt a világot egészítette ki a böngészőkben bővítményként megjelenő VRML-világ, valamint a böngészőoldali parancsfájlok és a Java-appletek lehetősége. Később az MS kidolgozta az appletek vetélytársaként is feltüntetett ActiveX-módszert.
- *A második nemzedékbeli webes alkalmazások* újdonsága a HTML-oldalak tartalmának dinamikus kialakítása volt. A Hálón szörföző emberek egyre unalmasabbnak és haszontalanabbnak tartották azokat a honlapokat, amelyek könyv módjára, teljes mértékben előre rögzített módon (statikusan) készültek. E módszerek legfőbb képviselői a CGI-programok, az SSI-k (Server Side Include). Érdekes kezdeményezés volt, hogy a webkiszolgálókat – hasonlóan más kiszolgálókhoz – API-val lássák el. Ennek legfőbb hátrányát az képezte, hogy minden webkiszolgáló más-más API-val rendelkezhetett. A CGI-programokat mai szemmel vizsgálva néhány hátrányuk azonnal szembetűnik: nehezen lehet bennük a HTTP-protokoll állapotfüggetlen jellegéből eredő korlátokat kiküszöbölni (például folyamat- vagy viszonykezelés), a CGI-parancsállományok esetenként lassúak és sok erőforrást igényelnek, mert minden CGI-kérésnél egy teljes értékű folyamat indul el. A pontosság kedvéért azonban azt is fontos megemlíteni, hogy például a Perl vagy a PHP-eszközök és az őket futtató környezet igyekszik mindent megtenni azért, hogy a fenti hátrányokat a lehető legjobban mérsékeljék.
- *A harmadik nemzedékbeli kiszolgálóalkalmazások* akkor születtek meg, amikor a fejlesztőeszközöket készítő cégek rájöttek arra, hogy az internetes (és belső hálózati) helyeket is érdemes lenne a hagyományos alkalmazáskészítő szemlélet jegyében fejleszteni. A végcél természetesen itt is az, hogy dinamikus HTML-, újabban XML-oldalakat hozzunk létre a böngésző ügyfél számára. Ennek a felfogásnak nyilvánvaló előnyei vannak. Használhatjuk a már ismert programozási nyelveinket és a vizuális fejlesztőeszközöket. Ennek az irányzatnak az egyik első megvalósítása a Microsoft Active Server Pages (ASP) szabványon alapuló megoldása volt, melynél az alkalmazásokat a Visual InterDev nevű eszközzel lehetett elkészíteni. Az ASP megoldás arra épül, hogy az ActiveX elemeket és az azokat működtető Visual Basic parancsfájlokat a kiszolgálóoldalon használja. Mi ennek a hátránya? Az MS-re oly jellemző megoldás: nagyszerű az ötlet, de a megoldás több részletében sem szabványos. Például a Visual Basic nyelv használata a szabványos C++ vagy Java helyett, a módszer – szerintem – csak windowsos környezetben működik rendesen, amit az OLE, az ActiveX és a Windows lehetőségeinek teljes kihasználásával lehet magyarázni. A hírek szerint az MS újabb stratégiája a „.NET”-elgondolás, amely mindjárt egy új nyelv bevezetésével indul (neve: C#). Vajon miért?

Ezután felmerül a kérdés, hogy létezik-e olyan webes fejlesztő és futtató környezet, amely tudja mindazt, amit az ASP, de felület- és gyártófüggetlen? A válasz igen. A megoldás elnevezése hasonlít az ASP nevére: JavaServer Pages (JSP). Itt szeretnék két dolgot megjegyezni:

➲ http://java.sun.com

1. A jelenlegi legelterjedtebb kiszolgálóoldali módszer még most is a Perl parancsállomány, ennek hátránya a CGI jellegű működésben rejlik.
2. Jelentősen terjed egy másik parancsnyelvi megoldás is, a PHP (PHP Hypertext Processor). A PHP egyszerűsége és nagyszerű alapötlete pont az, ami a JSP és ASP módszereket is jellemzi: készítsünk olyan HTML-oldalt, amelybe azok a kiszolgálóoldali programok vannak beágyazva (itt PHP-parancsfájlok), amelyek egy HTTP-kérés során lefutnak és az így dinamikusan kialakult HTML- (XML-) tartalom kerül át a böngészőbe. A PHP hátránya, hogy igazából ez is egy CGI módszer, néha szükségtelenül bonyolult (például ahány adatbázistípus, annyiféle API van az elérésükhöz).

A JSP és ASP rövid összehasonlítása

Térjünk vissza a JSP-re. Miért is jó ez? Vegyük sorra a párhuzamokat az ASP-vel:

- Az ASP kiszolgálóoldali ActiveX-objektumokat a Java osztályszerkezete fedi le. Ezeket az osztályokat éppen ezért servleteknek is nevezzük. Érdemes az üzleti logikát JavaBeanekben vagy – az IBM és a Sun kezdeményezése kapcsán – Enterprise JavaBeanekben (EJB) megfogalmazni. A JavaBean ugyanolyan elemalapú módszert jelent, mint ami a Delphi vagy Visual Basic (régebben VBX, manapság ActiveX) fejlesztői környezeteket is hatékonnyá teszi. A JSP abban ad nagy segítséget az alkalmazásfejlesztőnek, hogy ezeket a JavaBeaneket egyszerűen el tudja érni, azaz az alkalmazásréteg „tetején” a JSP végzi el a megjelenítő és adatbeviteli (HTML form feldolgozó) szolgáltatásokat, a JSP biztosítja a kapcsolattartást az ügyfélréteg és az alkalmazáslogika között.
- A Visual Basic parancsfájloknak szintén a Java program felel meg, amit ebben a környezetben néha scriptletnek is neveznek. Ki más is mozgósíthatná a .class fájlokban lévő servleteket, mint maga a Java nyelv? Ezen a téren az ASP és a JSP közötti hasonlatosság tökéletes. A JSP-oldalak az ASP-hez teljesen hasonló módon használják a <% ... %>, <%= ... %>, <%@ ... %> tagokat a kiszolgálóoldalon. Ezzel a JSP is tökéletesen megvalósítja az Active Documentnek nevezett megoldást. A különbség csak az, hogy a parancsfájl nyelve maga a Java. Itt felhívjuk a figyelmet egy nagyon fontos tényre: a .class fájlok dinamikus betöltése következtében a teljes Java-eszközkészletet használhatjuk a JSP-oldalakon. Ez azt jelenti, hogy az alkalmazást teljes egészében megírhatjuk Javában, és a HTML/XML-alapú felhasználói felületet kell csak JSP-ben megoldani. A JSP egyébként a servletek továbbgondolása kapcsán született meg. Régebben az SHTML (kiszolgáló-oldali HTLM) fájlok voltak azok, amelyekbe az applethez hasonlóan a <SERVLET code=….class> értékek… </SERVLET> tagok segítségével ágyazhattunk be egy servletet.
- Az ügyféloldali ActiveX-objektumoknak a Java appletek felelnek meg. Amiként az ügyfél- és a kiszolgálóoldali ActiveX objektumok is megvalósíthatnak ügyfél-, illetve kiszolgálókapcsolati lehetőséget, úgy lehetséges az applet és a servlet párbeszéde is.
- Az ASP COM/DCOM-nak a szabványos CORBA felel meg (a CORBA egy környezet- és nyelvfüggetlen protokoll megvalósítása).
- Az ASP OLE DB, ADO vagy ODBC adatbázis-elérési modellje helyett az objektumközpontú JDBC-t használhatjuk.

Ezen a ponton hagyjuk most abba az ASP és a JSP összehasonlítását. A JSP minden operációs rendszeren rendelkezésünkre áll, sőt legjobb megvalósításai a GNU felhasználási szerződése alá tartoznak, azaz beszerzésük ingyenes. (Vigyázat, ez nem a teljes tulajdonlási költséget jelenti!).

A JSP servlet működése

A JSP-oldalak első használata során azokból egy-egy servlet, azaz egy .class fájl keletkezik. A JSP célja tehát az, hogy ne kelljen a viszonylag részletgazdag servletek fejlesztésével foglalkoznunk, figyelmünket a dokumentumok kialakítására irányíthassuk (a beágyazott PHP parancsfájlhoz hasonlóan). Régebben a servletek meghívása hasonló volt a külső CGI parancsfájlok hívásához, azaz a cím így nézett ki: „http://gép/egy_servlet”. Később kialakult az SHTML-módszer, ennek használata a beágyazott PHP-hoz hasonló. Itt már a dokumentum szerkezete áll a figyelem középpontjában, mert a servletek ebbe vannak beágyazva. Jelenleg a JSP tekinthető ezen irányvonal csúcsának. A gondolat azért nagyszerű, mert a JSP-oldalakat egy webgrafikus is el tudja készíteni (ő úgy érzékeli a JSP-tagokat, mintha azok különleges HTML-tagok lennének), ugyanakkor a háttérben továbbra is a jól bevált servletek állhatnak. Itt jegyzem meg, hogy ez a felfogás tökéletes összhangban áll a mostanában viharos sebességgel terjedő XML-módszerrel.
A következőkben olyan JSP servletfejlesztő és -futtató programkörnyezet-kialakítást ismertetek, amely bármelyik operációs rendszeren megvalósítható és ebben a pillanatban a legkorszerűbb. A kialakítás ismertetését Linux operációs rendszerre írom le, de az alapelvek más operációs rendszereknél is teljesen hasonló. A harmadik részben pedig példán keresztül mutatom be e környezet használatát.

A fejlesztői és a futáskörnyezet kialakítása

Nézzük először is az általam használt hozzávalókat:

- Apache 1.3.14 kiszolgáló,
- Apache Jakarta-Tomcat 3.2.1,
- Sun Java 2 SDK Standard Edition for Linux,
- Borland Interbase v6.0 adatbázis-kiszolgáló,
- Borland InterClient JDBC meghajtó az Interbase eléréséhez.

A fenti programok közös jellemzője megbízható működésük, nagy cégek gyártják őket és a beszerzésük ingyenes. Kezdjünk neki kiszolgálónk kialakításának!

Az Apache telepítése

Az Apache kiszolgáló forráskódja a ➲www.apache.org helyről letölthető. A letöltendő fájl neve: apache_1.3.14.tar.gz.
A forráskódból telepített rendszereket a /opt könyvtár alá másoljuk:

```
$ cp apache_1.3.14.tar.gz /opt
```

Csomagoljuk ki:

```
$ cd /opt
$ tar -xvzf apache_1.3.14.tar.gz
```

Ennek hatására a forráskód a /opt/apache_1.3.14 könyvtárba kerül, erre – csupán a kényelem kedvéért – készítsünk közvetett hivatkozást:

```
$ ln -s apache_1.3.14 apache
```

Linux-rendszeren több módszer létezik arra, hogy a forráskódból hogyan készítsünk futtatható rendszert. A legelterjedtebb a configure nevű parancsállomány, ez feltérképezi Linux-rendszerünket és ahhoz illeszkedő Makefile állományt állít elő. Ezt használja az Apache is. Adjuk ki ennek megfelelően a `configure` parancsot:

```
./configure
            --sysconfigdir=/etc/httpd  \
            --datadir=/home/httpd  \
            --logfiledir=/var/log/httpd \
            -- disable-rule \
            --enable-shared=max \
            --enable-module=most
```

Ez a parancs olyan Makefile-t készít, amely előírja azt is, hogy hol lesz az Apache beállítási fájljának a helye (/etc/httpd), a webtartalom gyökere (/home/httpd) és a naplófájl. Továbbá engedélyezzük a futás közben betöltődő modulok használatát is.
A következő lépés a forráskód lefordítása a `make` parancs kiadásával. A fordítás után a bináris kód telepítése a `make install` paranccsal lehetséges. A futtatható Apache-rendszer a telepítés után a /usr/local/apache könyvtárban helyezkedik el.

A SUN JAVA 2 telepítése

A jdk-1.2.2-se.tar.gz fájlt (Java 2 Standard Edition) is másoljuk a /opt könyvtárba, majd a `tar -xvzf jdk-1.2.2-se.tar.gz` segítségével az ismertetett módon csomagoljuk ki. Ekkor létrejön egy /opt/jdk1.2.2 könyvtár, ahol a Java 2 rendszer található. Készítsünk erre is hivatkozást:

```
ln -s jdk1.2.2 jdk
```

Egy kicsit előretekintve módosítsuk a /etc/profájl fájlt, mert itt állítja be az általunk használt bash-héj környezeti változóit (Windowsban erre az autoexec.bat-ot használnánk).

```
# A /etc/profile végére ezt írjuk:
# A java home könyvtár
JAVA_HOME="/opt/jdk"
# A programok keresési útvonalának kiegészítése:
PATH=$JAVA_HOME/bin:$PATH
# A TOMCAT JSP és servlet szolgáltató helye
TOMCAT_HOME="/opt/tomcat"
# Az apache helye
APACHE_HOME="/usr/local/apache"
TCJ=/opt/tomcat/lib
JDKJ=/opt/jdk/jre/lib
CLASSPATH=.:$JDKJ/rt.jar
CLASSPATH=$CLASSPATH:$TCJ/servlet.jar:
      ➥$TCJ/jasper.jar:$TCJ/ant.jar:
      ➥$TCJ/servlet.jar:$TCJ/jasper.jar:
      ➥$TCJ/ant.jar:$TCJ/servlet.jar:
      ➥$TCJ/jasper.jar:$TCJ/ant.jar:$TCJ/jaxp.jar:
      ➥$TCJ/parser.jar:$TCJ/webserver.jar:
      ➥$JAVA_HOME/lib/tools.jar
export CLASSPATH, JAVA_HOME, TOMCAT_HOME,
APACHE_HOME
# A /etc/profile változtatás vége.
```

Ezekkel a változtatásokkal a Java fordító és futtató is meg fogja találni a .class fájlokat. Ezeket a .class fájlokat régebben a fájlrendszerben helyezték el, majd .zip fájlban tárolták (itt is benne volt a könyvtárszerkezet). Mostanában .jar fájlokat használnak, ennek zip a formátuma, de a jar Javában íródott. A Java fordító és futtató a .class fájlokat igény szerint, dinamikusan használja, ehhez azonban meg kell találni őket. A keresés a következőképpen zajlik: a program indító könyvtára, ha itt nincs, akkor a JAVA_HOME által kijelölt helyen lesz. Amennyiben ott sincs, akkor a CLASSPATH által megjelölt helyeken folytatódik a keresés.

Az Apache Jakarta-Tomcat telepítése

Az Apache-projekt webkiszolgálóját már régen kiegészítette servlethívási modullal, amit a mod_jserv.so (Apache_JSERV csomag) modul valósít meg. A JSP használatát először az erre épülő GNU JSP tette lehetővé. Ezt a párost váltotta fel az egységes Jakarta-Tomcat csomag, ami a servlet és JSP-lehetőségeket is magában foglalja. A csomagot szintén az Apache projekt ➲ www.apache.org helyéről lehet letölteni. A letöltendő fájl neve: jakarta-tomcat-3.2.1.tar.gz (bináris), illetve jakarta-tomcat-src-3.2.1.tar.gz (forráskód). Másoljuk be mindkét fájlt a megszokott /opt helyre, majd csomagoljuk ki a bináris csomagot a `tar -xvzf jakarta-tomcat-3.2.1.tar.gz` paranccsal, ennek hatására létrejön a /opt/jakarta-tomcat-3.2.1 könyvtár. Készítsünk rá hivatkozást:

```
ln -s /opt/jakarta-tomcat-3.2.1 tomcat
```

Emlékezzünk vissza, hogy a /etc/profile fájl TOMCAT_HOME változója erre a hivatkozásra mutat. A Tomcat telepítése ezzel a kicsomagolással lényegében befejeződött. Még annyit kell tenni, hogy a Tomcat-rendszer mod_jk.so nevű fájlját bemásoljuk a /usr/local/apache/libexec helyre. Itt tárolja az Apache a dinamikusan töltődő modulokat. A mod_jk.so csatolófelülete valósít meg az Apache számára. Ez azt jelenti, hogyha .JSP vagy servlet kérés megy az Apache felé, akkor az ezt kiutalja a mod_jk.so modul használatával a Tomcat felé. A Tomcat előállítja a dinamikus HTML-tartalmat, ezt az Apache szolgáltatja az ügyfél felé. Ahhoz, hogy az Apache tényleg így működjön, ezt meg kell mondani neki. Létezik a /opt/tomcat/conf könyvtárban egy „mod_jk.conf-auto” fájl, amit nem szabad szerkeszteni, mert úgy jó, ahogy van. A teendőnk csak annyi, hogy az Apache beállítófájljába a /etc/httpd/httpd.conf-ba utolsó sorként írjuk be a következőt:

```
include /opt/tomcat/conf/mod_jk.conf-auto
```

Ezzel az Apache jól fogja használni a kapcsolatot a Tomcathez.
A Tomcatet viszont még be kell állítani. Ezt a server.xml és a workers.properties fájlok módosításával tehetjük meg. Mindkettő a /opt/tomcat/conf könyvtárban helyezkedik el.

1. A workers.properties módosítása a következő sorok beírását, kijavítását jelenti:

```
# adjuk meg, hogy hol van a Tomcat
workers.tomcat_home=/opt/tomcat
# adjuk meg, hogy hol van a Java
workers.java_home=/opt/jdk
```

```
# adjuk meg, hogy a Linuxban ez a path határolójel
ps=/
# A Java virtuális gép helye
worker.inprocess.jvm_lib=/opt/jdk/jre/lib/i386/
classic/libjvm.so
```

2. A server.xml szerkesztése
Ha jó nekünk, hogy a JSP fájlok és a servletek a /opt/tomcat/webapps/root helyen vannak, akkor ezt a fájlt nem szükséges átírni.
Ezzel a Tomcat is működőképes lett. Hogyan indítsuk ezek után a webkiszolgálónkat?
Az első lépés a Tomcat indítása:

```
/opt/tomcat/bin/startup.sh.
```

Ez a parancsfájl is a kicsomagolt Tomcat része.
A második lépés az Apache indítása:

```
/usr/local/apache/bin/apachectl start
```

A webkiszolgáló leállítása a `/usr/local/apache/bin/apachectl stop`, majd a `/opt/tomcat/bin/shutdown.sh` parancspárral lehetséges.

A Borland Interbase adatbázis-kiszolgáló telepítése

A Borland szabaddá tette Interbase néven futó adatbázis-kiszolgálóját, ami tudását tekintve megközelítőleg az MS SQL kiszolgálóval egykategóriás, és webes célokra kiválóan alkalmazható.
A csomag telepítése rendkívül egyszerű. Az ...Interbase...6.0.tar.gz csomagot másoljuk a /opt helyre és a `tar -xvzf ...` paranccsal csomagoljuk ki. Ezután csak annyi a teendő, hogy a /etc/services fájlba beírjuk az új TCP/IP-szolgáltatást, azaz a fájlt a következő sorral kell kiegészíteni:

```
gds_db 3050/tcp
# hálózatos interbase
```

Ebből azt is látjuk, hogy az Interbase-t használó ügyfelek másik számítógépen is futhatnak, és amikor az adatbázis-kiszolgálót akarják használni, akkor ezt a 3050-es TCP-kapun tehetik meg. Ez hasonló az Oracle Net * protokolljához.
Az Interbase kiszolgálót a következő paranccsal lehet elindítani:

```
# az & hatására a folyamat háttérben fut.
/opt/interbase/bin/ibserver &
```

Az Interbase telepítésekor felkerült egy isql nevű program is, ami az adatbázis felé egy SQL parancsfelületet nyújt (hasonlóan, mint az Oracle *Plus). Indítsuk el „sysdba” felhasználóként, „masterkey” jelszóval:

```
isql -u sysdba -p masterkey
```

Ekkor egy „SQL>” készenléti jelet kapunk meg, ahonnan SQL-parancsokat adhatunk ki. Hozzunk létre egy új adatbázist az /opt/interbase/adatok könyvtárba:

```
SQL> create database
"/opt/interbase/adatok/webimi.gdb";
```

Az Interbase-adatbázisok egy-egy gdb kiterjesztésű fájlban találhatók (itt van minden: táblák, nézetek, tárolt eljárások, triggerek, indexek stb.).

A következő dbgen.sql parancsfájl egy táblát hoz létre ebben az adatbázisban:

```
# dbgen.sql
create table TELEFONOK
(
 azon    NUMERIC(10) NOT NULL,
 nev     VARCHAR(50),
 telefon VARCHAR(20),
 primary key( azon )
);
```

Ezt a parancsfájlt az isql `input` parancsával futtathatjuk:

```
input /opt/interbase/adatok/dbgen.sql;
```

Próbaként szúrjunk be egy sort ebbe a táblába:

```
Insert into TELEFONOK values(1, 'Nyiri Imre',
'123456789');
commit;
```

Lépjünk ki az isql-ből az `exit`; paranccsal.

Az Interbase Java JDBC meghajtó telepítése

A meghajtó az Interclient...tar.gz fájlban található, amit az eddigiek szerint másoljunk a /opt könyvtárba és csomagoljuk ki.
Ekkor létrejön a /opt/interclient_install_temp_dir könyvtár, lépjünk be, majd futtassuk le az ott lévő „install.sh” parancsfájlt. A parancsfájl azon kérdésére, hogy hova telepítse a JDBC meghajtót, adjuk meg a /opt/interclient könyvtárat. A telepítés után már csak annyi a teendő, hogy az interclient.jar nevű fájlt hozzáfűzzük a CLASSPATH-hoz a /etc/profile-ban. Ezzel kész az Interbase JDBC meghajtó telepítése. Próbáljuk ki, hogy működik-e! Írjunk ehhez Java programot.

```
//
// A program az előző pontban létrehozott
// adatbázist használja
// asz1.java, ahol asz=adatszolgáltató
//
import java.sql.*;
public class asz1
{
  public String ir()
  {
    String databaseURL =
              ➥"jdbc:interbase://localhost/opt/
              ➥interbase/adatok/webimi.gdb";
    String user = "sysdba";
    String password = "masterkey";
    String driverName =
              ➥"interbase.interclient.Driver";

    Driver d = null;
    Connection c = null;
    Statement s = null;
    ResultSet rs = null;

    // Driver load
    try
    {
     Class.forName
              ➥("interbase.interclient.Driver");
    }
    catch ( Exception e)
```

```
    {
      return "???";
    }

    // Kapcsolatlétrehozás
    try
    {
        c = DriverManager.getConnection
(databaseURL, user, password);
    }
    catch ( SQLException e)
    {
        return "???";
    }

    // auto commit hamisra állítása
    try
    {
        c.setAutoCommit (false);
    }
    catch (java.sql.SQLException e)
    {
        return "???";
    }

   // Egy lekérdezése
   try
      {
         s = c.createStatement();
        rs = s.executeQuery
         ("select NEV, TELEFON from TELEFONOK");
       ResultSetMetaData rsmd = rs.getMetaData();
       int cols = rsmd.getColumnCount();
       rs.next();
       return rs.getString("NEV");
      }
      catch ( SQLException e )
      {
            return "???";
      }
 }
} // end class
```

Most fordítsuk le a programot a `javac asz1.java` paranccsal, ennek eredményeként keletkezik az asz1.class fájl. Próbaképpen írjunk egy konzolalapú Java főprogramot, amely ezt az osztályt használja:

```
// imre.java
public class imre
{
  public static void main(String[] args)
  {
    System.out.print( new asz1().ir() );
  }
}
```

A program létrehoz egy új „asz1” típusú objektumot és meghívja annak az ir() tagfüggvényét, ami a telefonok tábla első sorának NEV mezőjét, tehát a „Nyiri Imre” karakterláncot adja vissza. A System.out.print() pedig kiírja ezt a konzolra. Működik tehát az adatbázis-kapcsolat. A programot a `java imre` paranccsal indíthatjuk el. Ennek hatására a „Nyiri Imre” szöveg kiíródik a konzolra.

Ezzel a fejlesztő, futtató környezet minden eleme a helyére került, és befejeztük a programok telepítését.

Az Apache – JSP környezet kipróbálása

A környezet kipróbálásához felhasználjuk az eddig létrehozott eredményeinket:

- A webimi.gdb adatbázist (benne a Telefonok táblát)
- A már elkészített asz1.java és asz1.class fájlokat

A jobb érthetőség kedvéért, valamint a servletek és a JSP-oldalak összehasonlíthatóságához elkészítünk egy olyan dinamikus HTML-oldalt létrehozó modult, amely olyan HTML-oldalt küld az ügyfeleknek, ami a napszaknak megfelelően köszön, majd kiírja a *Telefonok* tábla első sorának telefontulajdonosát.

A feladat JSP-alapú megoldása

Nos tehát, az általunk kialakított környezet működőképességének vizsgálatához készítsünk egy nagyon egyszerű JSP-oldalt, melynek a neve legyen elso.jsp. Ezt a böngészőből a „http://localhost/elso.jsp” sorral vagy egy erre mutató hivatkozással hívhatjuk meg.
Nézzük meg a JSP fájl tartalmát (a JSP fájlok text fájlok):

```
<%@ page import="java.util.Calendar" %>
<% if (Calendar.getInstance.get(Calendar.AM_PM)
       == Calendar.AM ) { %>
Kellemes délelőttöt kívánok! <br>
<% } else { %>
Kellemes délutánt kívánok! <br>
<% } %>
A telefontulajdonos neve: <%= new asz1().ir() %>
<br>
```

Az elso.jsp-re való hivatkozás után az Apache felméri, hogy ez JSP-oldal-e. A mod_jk csatoló segítségével megkéri a Tomcatet, hogy dolgozza fel ezt a fájlt. A Tomcat az elso.jsp fájlból a háttérben egy Java classt (servletet) készít, majd ezt lefuttatja a Java VM-mel. Itt van egy fordítási szakasz, ami csak az első alkalommal történik meg, utána már mindig a lefordított class fog futni. Ennek eredménye a következő dinamikus HTML lesz (csak amit a böngészőből látunk):

Kellemes délutánt kívánok!
A telefontulajdonos neve: Nyiri Imre

Ugye, milyen nagyszerű? Volt egy teljes Java alkalmazásunk az asz1.class fájlban (ezt egy kis túlzással egy JavaBeannek is nevezhetnénk), amit meghívtunk a JSP oldalról. Ugyanakkor az is látszik, hogy tetszőleges Java program írható be a JSP-oldalakra.

A feladat servletalapú megoldása

Természetesen most csak nagyon egyszerű megoldást fogunk látni, így aki a servleteket akarja tanulmányozni, annak ajánljuk a Sun servlet megvalósításait. Nézzük a programot:

```
// Importok
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
import java.net.*;
import java.util.Calendar;

//
// Ennek hívása a böngészőből:
// http://localhost/elso.class
//
```

```
public class elso extends HttpServlet
{
  // a servlet előkészítése
  public void init( ServletConfig conf ) throws
ServletException
  {
    super.init( conf );
  }

  // Szolgáltatása: html oldal előállítás
  //
  public void service( HttpServletRequest req,
                       // inputkérés
                          HttpServletResponse res )
                       // html outputhoz
  {
     String sz = (new asz1()).ir();
                       // a TELEFONOK táblából
     String udv; // üdvözlés

     if (Calendar.getInstance.get(Calendar.AM_PM)
                       == Calendar.AM )
     {
      udv = "Kellemes délelőttöt kívánok! <br>";
     }
     else
     {
      udv = "Kellemes délutánt kívánok! <br>";
     }

    res.SetContentType("text/html");
    ServletOutputStream out =
              res.getOutpitStream();
    out.println("<head><title>Az
              elso.class</title></head><body>");
    out.println( udv );
    out.println("A telefontulajdonos neve: ");
    out.println( sz ); out.println("</body>");
  } // end service

} // end elso class
```

Tekintettel arra, hogy a Tomcat is servletté fordítja a JSP-oldalt (a Tomcat work könyvtárában azt is mindig megnézhetjük, hogyan is néz ki ennek a JSP-ből előállított servletnek .java forrása), így a servletek kezelését is ismeri. Az azonban talán már mindenki előtt nyilvánvaló, hogy érdemes kihasználni azt, ahogy a JSP a servletet önműködően előállítja.

Záró gondolatok

Ez az írás csupán rövid ismertető volt arról, hogy miért is és hogyan használható a Java nyelv a kiszolgálóoldali programozáshoz. Van még néhány olyan kérdés, amit tisztázni kell, mielőtt valaki belevág módszer mélyebb használatába. Nézzük őket!

Milyen fejlesztőeszköz használható?

Az nyilvánvaló, hogy egyszerű Java fejlesztői környezet (Borland Jbuilder, Oracle Jdeveloper, IBM VisualAge for Java) nem elégséges, hiszen a .java, és .class fájlokon kívül még a következő nyersanyagokra is szükség van:

- HTML-szerkesztő,
- JSP-szerkesztő,
- CGI parancsfájlok írása-kezelése,
- Appletkészítő eszköz,
- HTML parancsfájlírást támogató eszköz (JavaScript, VB script),
- Grafikák, hangok stb. kezelése.

Ezenkívül ezeket az alapanyagokat egységes webalkalmazásként kell vezérelni. Ezt a feladatot az MS Visual InterDev remekül ellátja, de létezik az IBM gondozásában (az e-Business jegyében) egy másik eszköz is, ami ugyanilyen magas szinten tud webes alkalmazást kialakítani és vezérelni, sőt a JSP-beli támogatottsága egyértelműen jobb az InterDevnél: az IBM WebSphere Stúdió.
Az IBM VisualAge-ben úgy lehet kipróbálni a webes alkalmazást, hogy egy időben egyik ablakban látjuk a JSP-kódot, a mellette lévőben a JSP-ből előállított servlet kódját, a harmadik ablakban pedig magát a megvalósított HTML, WAP vagy VoiceXML stb. kódját.
Az így kialakított és kipróbált alkalmazás ezután bármilyen környezetben futtatható (Oracle Appserver, Apache + Tomcat, Websphere stb.).
Tanulási célra alkalmas fejlesztői környezet a Sun JSWDK 1.0 eszköz, mely ingyenesen letölthető.

A HTML-oldalak űrlapjainak feldolgozása

A HTML-űrlap-feldolgozás kihasználja a Bean/JSP előnyeit. Az mindenkinek kedvére való, ahogy a PHP egy „nev” nevű beviteli elemet $nev néven ér el. A JSP-ben sem bonyolultabb a helyzet. Egy html űrlap „nev” és „lakcim” nevű mezőjét a bean egy-egy tagfüggvényével tudjuk kezelni, ezeket célszerű getNev(), getLakcim(), setNev(), setLakcim() névre keresztelni, ezután a JSP-motor képes ezeket a mezőket önműködően kezelni. Ezt a folyamatot *introspection*-nek hívják és hihetetlen módon leegyszerűsíti az űrlapfeldolgozást.

A webhely grafikájának megvalósítása

A JSP-oldal lényegében egy HTML-oldal, szerkesztése, formázása gyakorlatilag bármilyen HTML-szerkesztőben történhet. A servletek baja az volt (hasonlóan a CGI parancsfájlokhoz), hogy a HTML-oldal külalakját – kód formájában – magukban hordozzák. Ha az alkalmazás kinézetén valami nem tetszik, akkor a servlet kódjába kell belenyúlni, azt újra kell fordítani és be kell vezetni. A valóságban azonban vannak olyan emberek, akik jól tudnak programozni és vannak olyanok, akik szép formát képesek alkotni. Ennek megfelelően biztosítani kell a külön munkavégzés lehetőségét. Természetesen úgy is fogalmazhatunk, hogy a JSP megkíméli a programozót a látvány programozásától, mert azt ezek után vizuálisan is elkészítheti. A programozók elkészítik a beaneket, a látványtervezők (grafikusok) pedig a JSP-oldalakat, amit ugyanúgy tudnak formázni, mintha HTML-oldalak lennének. A JSP/Bean szemléletű alkalmazás módosítása egyszerű: ha valami rossz a logikában, akkor annak kijavításához nem kell a látványtervező és fordítva.

Nyíri Imre
(inyiri@mol.hu) jelenleg a MOL Rt.-nél dolgozik. Informatikai vállalkozásában az Internet, a Linux és a Java programozás gyakorlati hasznosításával foglalkozik, ennek ellenére örök szerelme még mindig a C++. Kedveli a tudományos és a fantasztikus irodalmat, illetve filmeket (kedvencei: Solaris és 2001 – Űrodüsszeia). Szívesen sportol.

Kapcsolódó címek

- http://java.sun.com
- http://javasite.bme.hu
- http://www.servlets.com
- http://www.javasoft.com/products/jsp
- http://www.jspin.com